Chapter 33: The UPP Subscription File

UPP stands for **UNIX Product Poll**. It is a layer on top of **UPD** that can be used to facilitate the update of products on a local **UPS** node as new versions become available on a product distribution node. **UPP** is configured on the local node by subscription files, which we describe in this chapter. The functions **UPP** can be configured to perform on a local node include:

- notify the client of new and updated products on a specified distribution node
- perform product installations and updates
- install/update product dependencies and resolve chains to maintain integrity of main product
- delete old product versions

33.1 UPP Subscription File Header

The header of the **UPP** subscription file consists of lines of the form:

```
variable = value
```

in which the following variables may be defined:

Table 33.1.0-a:

`file`	Always set this to the value **`upp`**
`mail_address`	The email address where you want command output to be sent
`dist_node`	The node name of the product distribution node to query for new/updated products
`newprod_notify`	Set to **`T`** (True) if you want to be notified of brand new products; otherwise, leave it out or set it to any other value (e.g., **`F`**)

Table 33.1.0-a:

data_dir	The full path to the directory where you want **UPP** to maintain bookkeeping files. Each subscription file must have its own `data_dir`. `data_dir` must be writable when called from **cron**.
host_types	A list of flavors (i.e., **`-H`** options, see Chapter 25: *Generic Command Option Descriptions*) for which to do product surveys. It defaults to the value given by **`ups flavor`**.

Values here can have ${VARIABLE} strings, which are expanded from the environment. In particular:

Subscribe to changes in your local databases (e.g., if you want mail when someone updates products on your local system):

```
dist_node      = file://localhost${PRODUCTS}
```

Send mail to whoever runs the upp command:

```
mail_address      = ${USER}@fnal.gov
```

33.2 Stanzas

After the header, the **UPP** subscription file consists of one or more stanzas, each bracketed by the lines `begin` and `end`. The number of stanzas per file is not limited. A stanza cannot refer to multiple products, however there can be multiple stanzas for the same product (e.g., for treating different instances of the same product differently). Each stanza has three elements:

- identification of a product or particular instances of a product
- identification of the condition(s) for which you want **UPP** to perform the instructions you give it (done via an *action* statement)
- a list of instructions, or functions to perform, for each condition

33.2.1 Product Instance Identification

The following terms can be used for matching a new or updated product instance on the distribution node:

Table 33.2.1-a:

product	Product name
flavor	Product flavor

Table 33.2.1-a:

version	Product version
qualifiers	Product qualifiers
prod_dir	Product root directory
chain	Product chain

Within a stanza, *all* instances that match a given set of values will be operated on (in contrast to the standard **UPS** and **UPD** matching algorithms; see Chapter 27: *Product Instance Matching in UPS/UPD Commands*). You must at least specify the product name (the product name alone matches all instances), all further specification is optional and used to restrict the set of instances matched. Typically, only product and sometimes flavor are specified.

33.2.2 Conditions and Instructions

After identifying a product instance(s) within the stanza, you need to tell **UPP** what condition(s) to look for regarding the product, and what to do when the conditions are met. One or more `action = <value>` lines can be included to set conditions, each followed by a list of functions to perform.

Actions

In a subscription file, the **`action`** keyword can take the following values (indicating the condition):

Table 33.2.2-a:

`newversion`	A new version of the product is installed on the distribution node.
`<chain>`	The product is chained to *chain*, where *chain* can be current, test, or any other predefined or user-defined chain (see section 2.3.5 *Chains*). E.g., **`action = current`**

List of Functions

The functions that can be used under an `action = <value>` line currently take no arguments. All of the behavior is assumed to be defined by the local **UPD** configuration (described in Chapter 32: *The UPD Configuration File*) when **UPP** is invoked.

Table 33.2.2-b:

`notify`	Place a notice of the new product instance in the mailed output.
`install`	Install the subscribed product via **`upd install`**.
`delete`	Delete existing instance via **`ups undeclare -Y`**.
`reget`	Short for: delete, then reinstall
`update`	Update via **`upd update table_file:ups_dir`**.
`resolve`	Run any **`ups declare`** commands as necessary to make chains match so that parent product and dependencies run properly together.
`ups_configure`	Run the **`ups configure`** action for the product
`ups_installasroot`	Run the **`ups installasroot`** action for the product

33.3 Examples

33.3.1 Sample UPP Subscription File

```
FILE = upp
MAIL_ADDRESS = somebody@fnal.gov
DIST_NODE = fnkits.fnal.gov
DATA_DIR = /var/adm/upp
NEWPROD_NOTIFY = T
#
#  example  of  watching  for  new  releases  of  a  particular
product:
#
begin
    product = xntp
    flavor = SunOS+5

    action = newversion
       notify
end

#
# example of a product you want installed, but not chained,
when it goes current:
#
begin
    product = ximagetools
    flavor = SunOS+5

    action = current
           notify
           install
end

#
# example of tracking kits closely:
# * when a new version comes out we notify
# *  when  it  is  declared  or  modified  as  test  we  reget  it,
assuming the product
#   is allowed to have internal changes while in "test". We
"resolve" to have it
#   declared test here.
# * when it is declared current, we install it (which only
does something if we
```

```
#   don’t have it) and update it to catch re-issues of table
files,etc.  We
#   "resolve" to have it declared current here.
#
begin

product = exmh
flavor = SunOS+5

    action = newversion
           notify

    action = test
           notify
           reget
           resolve

    action = current
           notify
           install
           update
           resolve
end
```

A Second Example

This mails the user whenever **my_product** versions are made current on the local system.

```
file=upp
mail_address=${USER}@fnal.gov
dist_node=file://localhost${PRODUCTS}
data_dir=${HOME}/.upp/myproduct

begin
    product=my_product
    action=current
    notify
end
```

33.3.2 A Longer Annotated Example

Here is a sample **UPP** subscription file with one stanza. It is more comprehensive than a typical subscription file, illustrating the use of *all* the supported actions and functions. Explanations are provided line by line.

Table 33.3.2-a:

`file = upp`	This identifies the file as a **UPP** subscription file.
`mail_address = joe@fnal.gov`	Send mail notifications to *joe@fnal.gov.*
`dist_node = fnkits.fnal.gov`	Use `fnkits.fnal.gov` (the central Computing Division distribution node where the `KITS` database resides) as the **UPD** product distribution node to contact
`data_dir = /var/adm/upp`	Use `/var/adm/upp` as the **UPP** bookkeeping directory
`newprod_notify = T`	Yes, notify me of new products appearing on the **UPD** server node (i.e., in the `KITS` database).
`begin`	Begin a stanza.
`product = exmh`	Subscribe to **exmh**. In other words, perform the following actions on it and on its dependencies (the **exmh** flavors and versions remain unspecified in this example, therefore all instances are matched).
`action = newversion`	Define in the following lines one or more functions to perform when a brand new version of **exmh** appears in `KITS`.
`notify`	Send a notification message to *joe@fnal.gov*
`reget`	Remove (via **`ups undeclare -Y`**) and then reinstall (via **`upd install`**) the appropriate instance on the local node, and the necessary dependencies.
`resolve`	**`upd install`** has determined which `ups declare` commands need to be run so that all the chains match up properly for the dependencies to work; run these commands.

Table 33.3.2-a:

`action = current`	Define in the following lines one or more functions to perform when a version of **exmh** is chained to current in `KITS`.
`notify`	Send a notification message to *joe@fnal.gov*
`install`	Install the current instance in `KITS` (and its dependencies as necessary) on the local node
`resolve`	**`upd install`** has determined which `ups declare` commands need to be run so that all the chains match up properly for the dependencies to work; run these commands.
`action = deprecated`	Define in the following lines one or more functions to perform when a version of **exmh** gets deprecated (i.e., chained to a user-defined chain of "deprecated") in `KITS`. This is included to illustrate the use of user-defined chains.
`notify`	Send a notification message to *joe@fnal.gov*
`delete`	Delete the instance on the local node via **`ups undeclare -Y`**.
`end`	End stanza. (Additional stanzas may be included in the same file; use `begin` and `end` to bracket each one.)